Technical Report - Spam Detection

Lennart Kasserra

Link to Repository

Introduction

Spam emails are a very common and well-known nuisance: they often contain phishing attempts, malware, and scams that can compromise personal data and financial security. While traditional rule-based filters have long struggled to adapt to evolving spam tactics, machine learning models can dynamically learn patterns from data and are thus more robust. The goal of this project was to build a model that could reliably classify emails as either "spam" or "no spam" ("binary classification task"), and to evaluate whether a Deep Learning approach was preferrable to using "traditional" models. The data was provided via the UC Irvine machine learning repository.

Overall, my findings indicate that a Deep Learning approach is not neccessary: a properly tuned Random Forest classifier outperformed a deep Neural Network on all evaluated metrics.

Analysis

The data contained 57 features, and one column denoting the class label (either spam or no spam). Most features were related to the presence and frequency of certain words or special characters, or excessive capitalization. Many of the features were very sparse and had heavily right-skewed distributions (more on this in the code). The classes were slightly imbalanced, with fewer spam than non-spam emails. I further ran a principal component analysis (PCA) to see if we could find some patterns in a transformed representation of the data.

After investigation, I decided on my preprocessing steps: despite class imbalance not being too bad, I decided to use synthetic minority class oversampling (SMOTE) to generate synthetic examples of spam emails and equalize the number of training examples per class. I further log-transformed and normalized all features, dropped highly correlated features ($r > 0.9$ or $r < -0.9$) and features with near-zero variance.

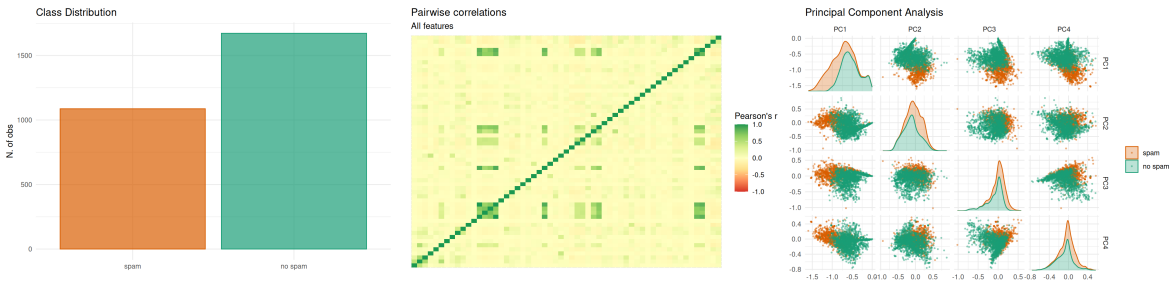


Figure 1: Exploratory data analysis: investigating class labels, feature correlations & a principal component analysis (training set only; training-validation-testing split was stratified on class label).

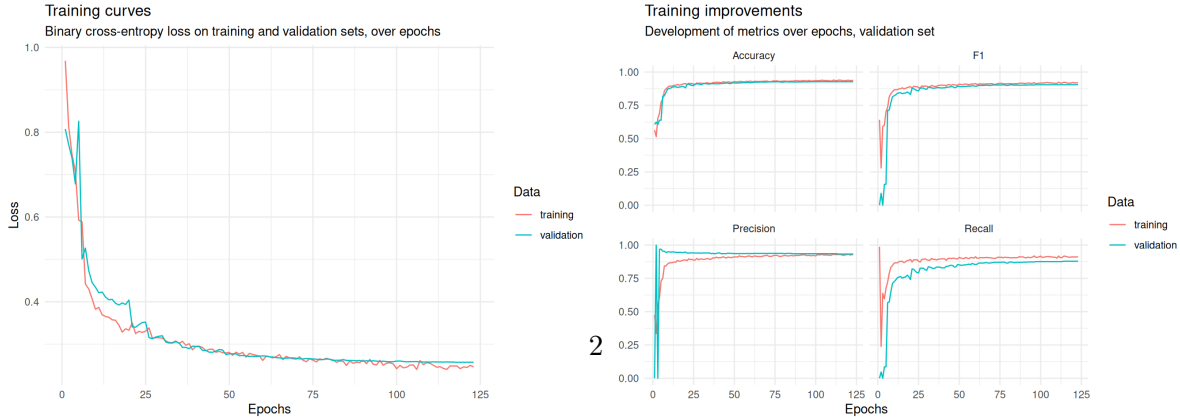
Methods

The main model is a deep Neural Network with four hidden layers. The hidden layers had 128, 64, 32 & 16 neurons respectively, all using the Rectified Linear Unit (ReLU) activation function and with dropout applied after every layer. The output layer was a one-neuron layer with sigmoid activation function, appropriate for this sort of binary classification task. Hyperparameters are shown in the table below.

Hyperparameter	Value
Learning rate	0.001 (scheduled)
Optimizer	Adam
Dropout Rate	0.25
Batch size	32
Epochs	max. 250 (early stopping)
L2 Regularization	0.001

Figure 2: Hyperparameters

Learning rate scheduling was set up to reduce the learning rate by a factor of 0.8 after three epochs of patience[1] ("Reduce on Pleateau"). Early stopping would kick in after validation loss has not decreased for five epochs in a row. Binary cross-entropy loss was used as the loss function for training.



(a) Loss curves

(b) Training curves for different metrics

Figure 3: Training process of the deep Neural Network.

[1] I.e. after validation performance did not increase for three successive epochs. This is meant to stabilize training & help the model settle into minima.

To select a proper competitor to compare the deep Neural Network against, I evaluated three "traditional" models: a penalized Logistic Regression model, a Gaussian Naive Bayes classifier, and a Random Forest. For all models, I tuned their hyperparameters using regular grid search with ten-fold cross validation [2], and then compared their performance on the training set to that of the Neural Network. Results can be found in figure 4.

I decided that for the given application - detecting spam - false positives (i.e. falsely labeling genuine emails as spam, and thus potentially removing them from the inbox) were more costly or undesirable than the annoyance of having an occasional spam email slip through, and thus decided that precision ("what proportion of emails classified as spam were actually spam") was the most crucial metric.

The best best Random Forest classifier's performance across the ten folds was roughly on par with that of the neural network on the training set in terms of precision, so I decided that the Random Forest maxed out for precision would become the main competitor.

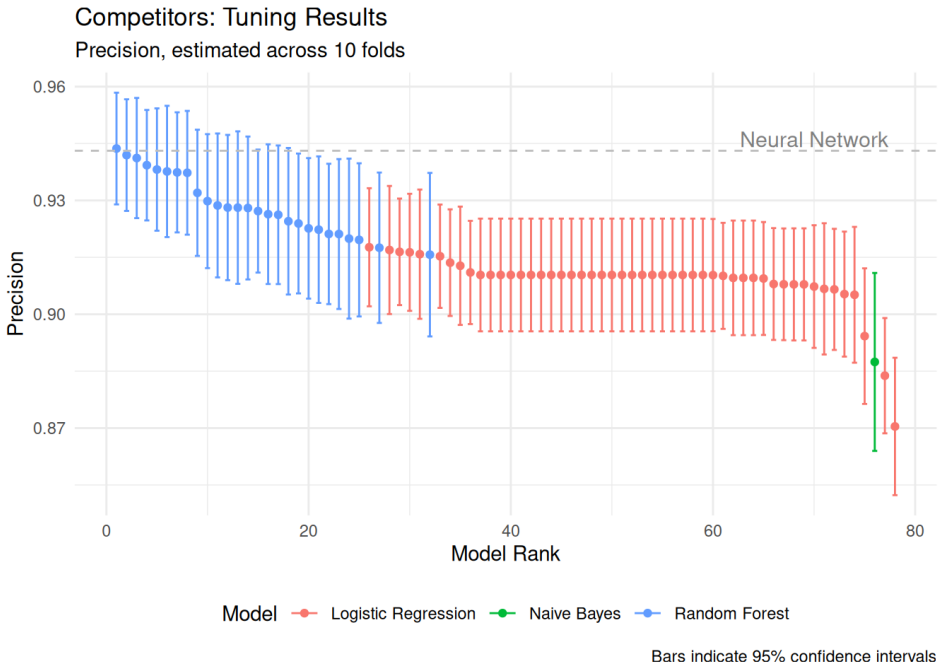


Figure 4: Ranking tuned potential competitors by precision achieved across ten folds.

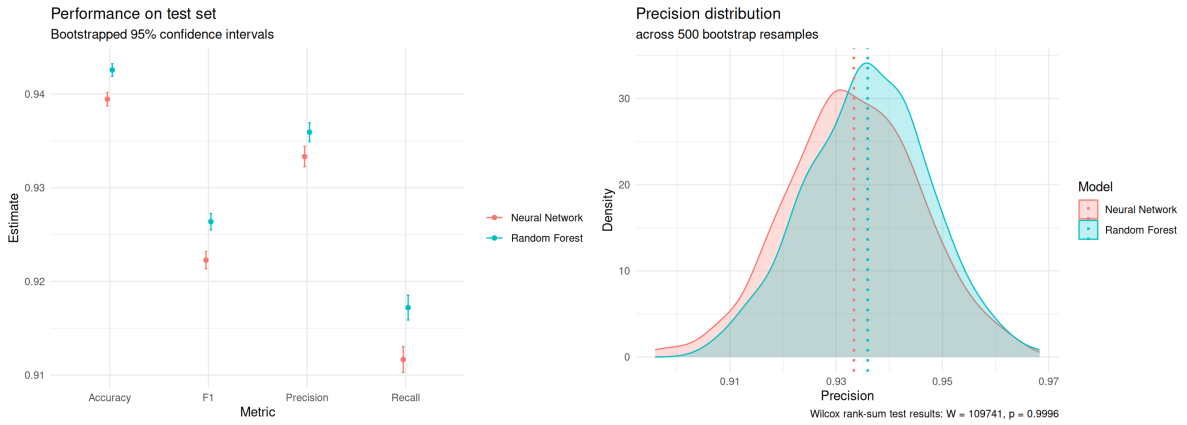
[2] Except for the Gaussian Naive Bayes, where I did not tune the smoothing parameter & just used raw probabilities instead (the smoothing should not have any effect in this setting).

Results

The deep Neural Network was outperformed by a tuned Random Forest model on all metrics when evaluated on the test set. Crucially, it delivered lower precision than the Random Forest, but also fell behind on accuracy and f1-score. While the performance differences are at best marginal, when combined with the explainability that Random Forest offers over a deep Neural Network this is a solid case for using a "traditional" over a Deep Learning approach. Below are the metrics computed on the test set, both as point estimates and with bootstrapped 95% confidence intervals.

Model	Precision	Recall	F1	Accuracy
Neural Network	0.932	0.912	0.922	0.939
Random Forest	**0.935**	**0.917**	**0.926**	**0.942**

Figure 5: Test metrics for the Neural Network and the Random Forest.



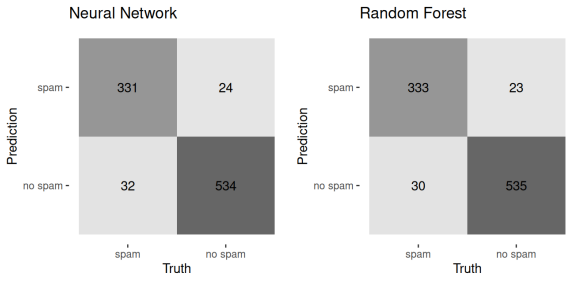
(a) Test metrics with bootstrapped 95% confidence intervals

(b) Distribution of precision estimates across resamples by model

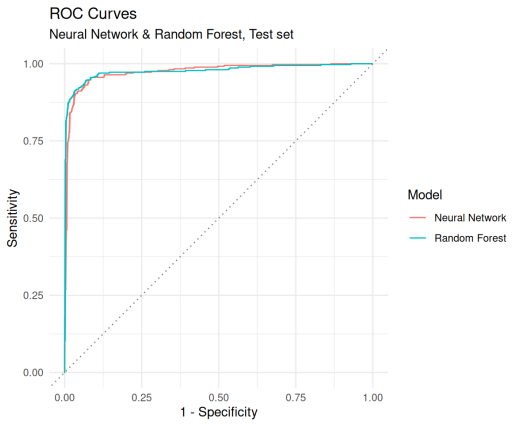
Figure 6: Bootstrap results across 500 resamples.

Additionally, I examined the models' "confidence" in their classifications by comparing their outputted class probabilities when being correct versus when being wrong. On average, the Neural Network is also overconfident in misclassifications[3], which might hint at a deeper structural problem. However, examining the ROC curves and the confusion matrices shows just how close together the two models are in terms of performance: the differences in test performance are driven by a very small number of observations.

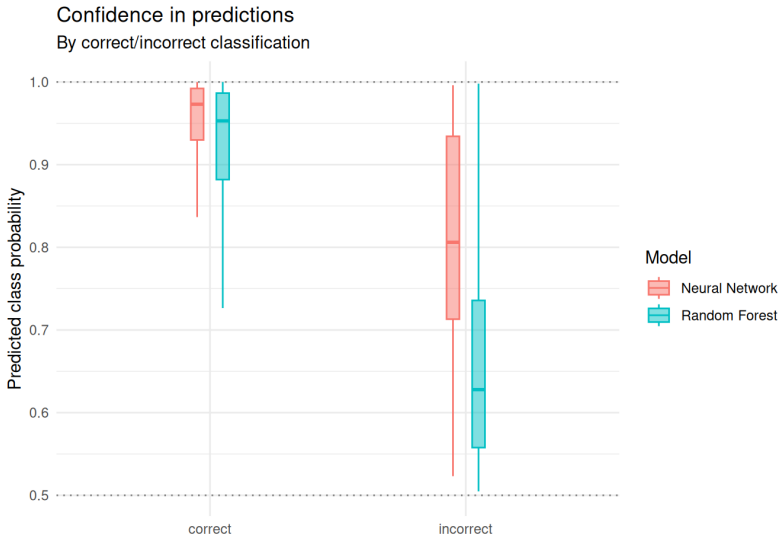
[3]A Wilcoxon rank-sum test confirms that the predicted probabilities for misclassified instances were significantly higher for the Neural Network compared to the Random Forest ($W = 2080$, $p < 0.01$, i.e. their distribution is significantly shifted).



(a) Confusion matrices



(b) Receiver-operating characteristic (ROC) curves.



(c) Model confidence in predictions: predicted class probabilities for right and wrong classifications.

Figure 7: Test performance examined in more detail.

Overall, a Deep Learning approach is not neccessary for this use case, and given this data.

Reflection

I encountered the biggest difficulties trying to set up a proper environment for Deep Learning with R. While I got it to work eventually using the `keras3`-package, parts of it - e.g. using CUDA - would not work properly until the very end, despite the very same GPU being accessible from Python. For this application, this was not a serious shortcoming as the model was not too complex and the amount of data was not too big, but for a more sizeable project this could have been a serious showstopper.

Regarding potential improvements to the current approach, it would have been tempting to try out more "traditional" models (e.g. gradient boosted trees), and to explore a larger search space when tuning hyperparameters. Further, dropping highly correlated and all near-zero variance predictors in preprocessing significantly reduces the number of available features and may have been a very drastic step; examining how the models would perform in the presence of these extra predictors would also be interesting, as neither Neural Networks nor Random Forest should struggle with the presence of these.

It might also have been fruitful to explore how smaller or more shallow Neural Networks compare to the deep model.